

[View](#) [Sync](#) [Diff Revs](#) [Diff vs Client File](#) [Describe Changelist](#) [Help](#)[OK](#)

Revision	File Name	Changelist	Date	Description
----------	-----------	------------	------	-------------

3	//depot/m...	2357	2000/01/14	Merge in calendar sync changes from Continuous
2	//depot/m...	2265	2000/01/13	Apply 2.0 files
1	//depot/m...	2100	2000/01/10	Add redirector to P4

Actions

Display Branching History

```

5  /*****
   MAPIMailbox.cpp
   (C) 1997 Research In Motion Ltd.
   *****/

#pragma warning( disable:4201 )
#pragma warning( disable:4514 )

#include "MAPIMailbox.h"

#include <assert.h>
#include <stdio.h>

#include "MAPIMailboxNotify.h"
#include "MailboxManager.h"
#include "globals.h"
#include "rimmessage.h"
#include "rimcalendar.h"
#include "rimstream.h"
#include "debug.h"
#include "Mutex.h"
#include "timeconversion.h"
#include "ICSAgent.h"
#include "CallCSAgent.h"
#include "icalformat.h"
#include "calsyncstate.h"

Mutex g_DebugMutex;

// Recurrence blob related structures and constants

35 #define _SECOND ((_int64) 100000000)
   #define _MINUTE (60 * _SECOND)
   #define _HOUR (60 * _MINUTE)
   #define _DAY (24 * _HOUR)

40 const USHORT TYPE_DAILY = 0x200A;
const USHORT TYPE_WEEKLY = 0x200B;
const USHORT TYPE_MONTHLY = 0x200C;
const USHORT TYPE_YEARLY = 0x200D;

```

```

11830         _cleanup:
11831         try
11832         {
11833             // resource cleanups
11834             if( lpStreamBody ) {
11835                 lpStreamBody->Release();
11836             }
11837
11838             return( rc );
11839         }
11840         catch( ... )
11841         {
11842             ReportException(mailbox_fcn);
11843             return( rc );
11844         }
11845         // Exception-handling
11846     }
11847
11848     /** RIMEventtoCDOAppointment ***
11849     // Purpose: Copies the Event properties into the corresponding members of the provided
11850     //           CDO appointment object.
11851     // Parameters:
11852     // Returns:
11853     //
11854     bool MAPIMailbox::RIMEventtoCDOAppointment( RIMEvent *pEvent, AppointmentItemPtr &spAppointmentItem )
11855     {
11856         static const char mailbox_fcn[] = "RIMEventtoCDOAppointment";
11857
11858         assert( pEvent );
11859
11860         RecipientsPtr      spAppointmentRecipients;
11861         RecipientPtr       spAppointmentAttendee;
11862         RecurrencePatternPtr spAppointmentRecurrence;
11863
11864         RIMEvent::FrequencyType ApptFrequency;
11865         bool fInstanceRecurrence = false;
11866         time_t tStartTime;
11867         FILETIME ftStartTime;
11868         SYSTEMTIME stStartTime;

```

```

11870     DOUBLE      dStartTime;
11871     time_t      tEndTime;
11872     FILETIME    ftEndTime;
11873     SYSTEMTIME stEndTime;
11874     DOUBLE      dEndTime;
11875     long        lAppointmentRefId = 0;
11876     long        lParentRefId = 0;
11877     long        lDeviceSeqNum = 0;
11878     bool        rc = true;
11879
11880     try
11881     {
11882         /* Get the UID (our ref Id) */
11883         RIMEvent::UIDProperty* pAppointmentUID;
11884         if( !pEvent->GetProperty( &pAppointmentUID ) )
11885         {
11886             DebugLog::Printf( DebugLog::LOG_WARNING,
11887                             "MAPIMailbox::RIMMeetingtoMAPIMeeting - Getting UIDProperty from RIMCalendar failed for %s.",
11888                             m_Name );
11889
11890             rc = false;
11891             goto _cleanup;
11892         } // if
11893         lAppointmentRefId = pAppointmentUID->GetUID();
11894
11895         /* Get the RelatedTo Id */
11896         RIMEvent::RelatedToIDProperty* pAppointmentRelatedTo;
11897         if( !pEvent->GetProperty( &pAppointmentRelatedTo ) )
11898         {
11899             lParentRefId = pAppointmentRelatedTo->GetRelatedToID();
11900
11901             /* Get the Sequence # */
11902             RIMEvent::SequenceProperty* pAppointmentSequenceNum;
11903             if( !pEvent->GetProperty( &pAppointmentSequenceNum ) )
11904             {
11905                 DebugLog::Printf( DebugLog::LOG_WARNING,
11906                                 "MAPIMailbox::RIMMeetingtoMAPIMeeting - Getting SequenceProperty from RIMCalendar failed for %s.",
11907                                 m_Name );
11908
11909                 rc = false;
11910                 goto _cleanup;
11911             } // if
11912             lDeviceSeqNum = pAppointmentSequenceNum->GetSequenceNumber();

```

```

11915 /* Get the Summary */
RIMEvent::SummaryProperty* pAppointmentSummary;
if( pEvent->GetProperty( &pAppointmentSummary ) )
{
    spAppointmentItem->PutSubject( _variant_t(pAppointmentSummary->GetSummary()) );
}
/* Get the Location */
RIMEvent::LocationProperty* pAppointmentLocation;
if( pEvent->GetProperty( &pAppointmentLocation ) )
{
    spAppointmentItem->PutLocation( _variant_t(pAppointmentLocation->GetLocation()) );
}
/* Get the Description */
RIMEvent::DescriptionProperty* pAppointmentDescription;
if( pEvent->GetProperty( &pAppointmentDescription ) )
{
    RIMStream DescriptionStream = pAppointmentDescription->GetDescription();
    DescriptionStream.Seek(); // Rewind
    unsigned int BodyLength = pAppointmentDescription->GetAvailableSize();
    unsigned char *pBodyText = new unsigned char[ BodyLength+1 ];
    memset(pBodyText, 0, BodyLength);
    DescriptionStream.Read(pBodyText, BodyLength);
    pBodyText[BodyLength] = '\0';
    _bstr_t bstrBody((const char *)pBodyText);
    spAppointmentItem->PutText(bstrBody);
    delete []pBodyText;
}
else
{
    // Set a blank message body
    spAppointmentItem->PutText(_bstr_t(""));
} //if

11945 /* Get the Trigger */
RIMEvent::TriggerProperty* pAppointmentTrigger;
if( pEvent->GetProperty( &pAppointmentTrigger ) )
{
    spAppointmentItem->PutReminderSet( _variant_t(1L, VT_I4));
    spAppointmentItem->PutReminderMinutesBeforeStart( _variant_t(pAppointmentTrigger->GetTrigger()) );
}
/* Get the Date Time Start */
RIMEvent::DateTimeStartProperty* pAppointmentDateTimeStart;
if( pEvent->GetProperty( &pAppointmentDateTimeStart ) )

```

MAPImailbox.cpp

```

11955 {
    DebugLog::Printf( DebugLog::LOG_WARNING,
        "MAPImailbox::RIMMeetingtoMAPIMeeting - Getting DateTimeStartProperty from RIMCalendar failed for %s.",
        m_Name );
    rc = false;
    goto _cleanup;
} // if
// The long road to convert this to a variant time
tStartTime = pAppointmentDateTimeStart->GetTime();
UnixTimeToFileTime(tStartTime, ftStartTime);
FileTimeToSystemTime(&ftStartTime, &stStartTime);
SystemTimeToVariantTime(&stStartTime, &dStartTime);
spAppointmentItem->PutStartTime( _variant_t(dStartTime, VT_DATE) );
/* Get the DateTimeEnd */
RIMEvent::DateTimeEndProperty* pAppointmentDateTimeEnd;
if( !pEvent->GetProperty( &pAppointmentDateTimeEnd ) )
{
    DebugLog::Printf( DebugLog::LOG_WARNING,
        "MAPImailbox::RIMMeetingtoMAPIMeeting - Getting DateTimeEndProperty from RIMCalendar failed for %s.",
        m_Name );
    rc = false;
    goto _cleanup;
} // if
// The long road to convert this to a variant time
tEndTime = pAppointmentDateTimeEnd->GetTime();
UnixTimeToFileTime(tEndTime, ftEndTime);
FileTimeToSystemTime(&ftEndTime, &stEndTime);
SystemTimeToVariantTime(&stEndTime, &dEndTime);
spAppointmentItem->PutEndTime( _variant_t(dEndTime, VT_DATE) );
/* Get the Attendee(s) */
RIMEvent::AttendeeProperty* pAppointmentAttendee;
spAppointmentRecipients = spAppointmentItem->GetRecipients();
pEvent->GetProperty( &pAppointmentAttendee );
while( pAppointmentAttendee )
{
    _bstr_t bstrAddress = "SMTP:";
    bstrAddress += _bstr_t(pAppointmentAttendee->GetAddress());
    spAppointmentRecipients->Add( _variant_t(pAppointmentAttendee->GetDisplayName()), _variant_t(bstrAddress), _variant_t((long)(CdoTo) ) );
    pEvent->GetNextProperty( &pAppointmentAttendee );
} // while
/* Get the Recurrence Pattern, if any */
RIMEvent::RecurrenceRuleProperty* pAppointmentRecurrence;
if( pEvent->GetProperty( &pAppointmentRecurrence ) )

```

```

{
    spAppointmentRecurrence = spAppointmentItem->GetRecurrencePattern();
    // Following items set no matter what the recurrence type is
    spAppointmentRecurrence->PutOccurrences(_variant_t(pAppointmentRecurrence->GetCount()));
    spAppointmentRecurrence->PutInterval(_variant_t(pAppointmentRecurrence->GetInterval()));
    spApptFrequency = pAppointmentRecurrence->GetFrequency();
    // Items that may or may not be present based on recurrence type
    if( pAppointmentRecurrence->GetDay() != RIMEEvent::UNDEFINED_DAY )
    {
        spAppointmentRecurrence->PutDayOfWeekMask(_variant_t(pAppointmentRecurrence->GetDay()));
    }
    if( pAppointmentRecurrence->GetMonth() != RIMEEvent::UNDEFINED_DAY )
    {
        spAppointmentRecurrence->PutMonthOfYear(_variant_t(pAppointmentRecurrence->GetMonth()));
    }
    if( pAppointmentRecurrence->GetMonthDay() != 0 )
    {
        spAppointmentRecurrence->PutDayOfMonth(_variant_t(pAppointmentRecurrence->GetMonthDay()));
    }
    if( pAppointmentRecurrence->GetSetPosition() != 0 )
    {
        fInstanceRecurrence = true;
        spAppointmentRecurrence->PutInstance(_variant_t(pAppointmentRecurrence->GetSetPosition()));
    }
    switch( ApptFrequency )
    {
        case RIMEEvent::DAILY:
            spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeDaily));
            break;
        case RIMEEvent::WEEKLY:
            spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeWeekly));
            break;
        case RIMEEvent::MONTHLY:
            if( fInstanceRecurrence )
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeMonthlyNth));
            else
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeMonthly));
            break;
        case RIMEEvent::YEARLY:
            if( fInstanceRecurrence )
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeYearlyNth));
            else
                spAppointmentRecurrence->PutRecurrenceType(_variant_t((long)CdoRecurTypeYearly));
    }
}

```

```

12045         break;
            } // switch
        } // if
    }
    catch ( _com_error &e )
    {
        _bstr_t bstrSource(e.Source());
        _bstr_t bstrDescription(e.Description());
        DebugLog::Printf( DebugLog::LOG_INFORMATIONAL,
            "*** CDO *** MAPIMailbox::MAPIMailbox( ) - Code = %08lx, Code meaning = %s, Source = %s, Description = %s.",
                e.Error(), e.ErrorMessage(), (LPCTSTR) bstrSource, (LPCTSTR) bstrDescription);

        rc = false;
    }
    catch( ... )
    {
        rc = false;
    }
    _cleanup:
    return( rc );
}

12060 // *** Send ***
// Purpose:
// Parameters:
//
bool MAPIMailbox::Send( RIMessage* pMessage, LPMESSAGE* ppMAPIMessage, bool bDeleteAfterSubmit,
    int origRefId, bool insert_text, bool insert_attach, FOLDERS eFolder )
{
    static const char mailbox_fn[] = "Send(ppMAPIMessage)";
    assert( pMessage );
    assert( ppMAPIMessage );

12075     LPMESSAGE     pOrigMessage = 0;
    HRESULT         hResult;
    bool            bRetCode = false;

    try
    {
        //DebugLog::Printf( DebugLog::LOG_DEBUG, "*** MAPI *** Sending message from %s", m_Name );
        if ( origRefId != 0 )

```



```

14150 {
    static const char mailbox_fcn[] = "TransferAppointmentReceivedNotification";
    EntryID *pEntry = new EntryID( m_pSession, EntryIDValue , EntryIDSize );
    assert( pEntry );
    // Notify user control.
    m_pControl->AppointmentModifiedNotification( pEntry );
    return;
}

14155 void MAPIMailbox::TransferAppointmentDeletedNotification( unsigned long SourceKeyIDSize, unsigned char * SourceKeyIDValue )
{
    static const char mailbox_fcn[] = "TransferAppointmentReceivedNotification";
    EntryID *pSourceKey = new EntryID( 0, SourceKeyIDValue , SourceKeyIDSize );
    assert( pSourceKey );
    // Notify user control.
    m_pControl->AppointmentDeletedNotification( pSourceKey );
    return;
}

14160 // *** Synchronize ***
// Purpose:
// Parameters: pAppointment = RIM calendar
// Returns: true if appointment sync'd successfully. Otherwise, false.
//
14165 bool MAPIMailbox::Synchronize( RIMCalendar* pAppointment )
{
    static const char mailbox_fcn[] = "Synchronize";
    #ifndef PERSONAL_BUILD // No calendar support for desktop redirector
    ResetLastError();
    LPMAPITABLE pContentsTable = 0;
    LPSRowSet pRows = 0;
    LPMESSAGE pRootAppointment = 0;
    LPMAPITABLE pAttachTable = 0;
    LPATTACH pAttachment = 0;
    14170
    14175
    14180
    14185
    14190

```

```

LPSRowSet      pAttachRows = 0;
LPSBinary      pbinStateSourceKey = 0;

FolderPtr      spCalendarFolder;
AppointmentItemPtr spAppointmentItem;
MessagesPtr    spCalendarMessages;
FieldsPtr      spAppointmentFields;
FieldPtr       spAppointmentSourceKey;

variant_t      viAppointmentSourceKey;
SRestriction    sSourceKeyRes;
SPropValue      spvSourceKey;
HRESULT         hResult;
bool            bResult = false;

const ULONG     nAppointmentProps = 7; // # of appointment properties

SizedSPropTagArray( nAppointmentProps, sptaAppointmentProps ) =
{ nAppointmentProps,
  {
    PR_ENTRYID,
    PR_SOURCE_KEY,
    PROP_TAG(PT_LONG, 0xE23),
    PR_RIM_CAL_APPT_DURATION,
    PR_RIM_CAL_START_DATETIME,
    PR_RIM_CAL_END_DATETIME,
    PR_LAST_MODIFICATION_TIME
  }
};

const ULONG     nAttachProps = 5; // # of attach properties

SizedSPropTagArray( nAttachProps, sptaAttachProps ) =
{ nAttachProps,
  {
    PR_ATTACH_LONG_FILENAME,
    PR_ATTACH_FILENAME,
    PR_ATTACH_MIME_TAG,
    PR_ATTACH_NUM,
    PR_ATTACH_METHOD
  }
};

```

```

14235 try
14236 {
14237     RIMCalendar::MethodProperty* pAppointmentMethod = 0;
14238     RIMCalendar::CalendarEventProperty* pCalendarEvent = 0;
14239     RIMEvent
14240     long
14241     long
14242     long
14243     time_t
14244     long
14245     FILETIME
14246     FILETIME
14247     FILETIME
14248     FILETIME
14249     SYSTEMTIME
14250     variant_t
14251     long
14252     long
14253     bool
14254     bool
14255     bool
14256
14257     // Determine the request method: PUBLISH or CANCEL
14258     // Calendar method
14259     if( !pAppointment->GetProperty( &pAppointmentMethod ) )
14260     {
14261         DebugLog::Printf( DebugLog::LOG_WARNING,
14262             "MAPIMailbox::Synchronize - Getting MethodProperty from RIMCalendar failed for %s.",
14263             m_Name );
14264         bResult = false;
14265         goto __end;
14266     }
14267
14268     // Loop over VEVENT components contained in the calendar object
14269     if( !pAppointment->GetProperty( &pCalendarEvent ) )
14270     {
14271         DebugLog::Printf( DebugLog::LOG_WARNING,
14272             "MAPIMailbox::Synchronize - Getting CalendarEventProperty from RIMCalendar failed for %s.",
14273             m_Name );
14274         bResult = false;
14275         goto __end;
14276     }

```

```

14280 while( pCalendarEvent )
{
    CalendarVevent = pCalendarEvent->GetCalendarEvent();
    // Get the CDO calendar folder object.
    if( spCalendarFolder == 0 )
    {
        spCalendarFolder = m_spCDOSession->GetDefaultFolder((long)CdoDefaultFolderCalendar);
    }

    // Find state info, if it exists, for this item using ref id
    lApptRefId = 0;
    lApptParentRefId = 0;
    lApptDeviceSeqNum = 0;
    spAppointmentItem = 0;
    spAppointmentFields = 0;
    spAppointmentSourceKey = 0;
    if( pbinStateSourceKey )
    {
        MAPIFreeBuffer( pbinStateSourceKey );
        pbinStateSourceKey = 0;
    } //if

    /* Get the UID (our ref Id) */
    RIMEEvent::UIDProperty* pAppointmentUID;
    if( !CalendarVevent.GetProperty( &pAppointmentUID ) )
    {
        DebugLog::Printf( DebugLog::LOG_WARNING,
            "MAPIMailbox::Synchronize - Getting UIDProperty from RIMCalendar failed for %s",
            m_Name );
        bResult = false;
        goto __end;
    }
    lApptRefId = pAppointmentUID->GetUID();

    /* Get the RelatedTo Id */
    RIMEEvent::RelatedToIDProperty* pAppointmentRelatedTo;
    if( CalendarVevent.GetProperty( &pAppointmentRelatedTo ) )
    {
        lApptParentRefId = pAppointmentRelatedTo->GetRelatedToID();
        RIMEEvent::RecurrenceIDProperty* pRecurrenceID;
        if( CalendarVevent.GetProperty( &pRecurrenceID ) )

```

```

14320 {
14321     tInstanceDate = pRecurrenceID->GetRecurrenceID();
14322     UnixTimeToFileTime( tInstanceDate, ftInstanceDate );
14323 }
14324 }
14325
14326 /* Get the Sequence # */
14327 RIMEEvent::SequenceProperty* pAppointmentSequenceNum;
14328 if( !CalendarVevent.GetProperty( &pAppointmentSequenceNum ) )
14329 {
14330     DebugLog::Printf( DebugLog::LOG_WARNING,
14331         "MAPImailbox::Synchronize - Getting SequenceProperty from RIMCalendar failed for %s",
14332         m_Name );
14333     bResult = false;
14334     goto __end;
14335 }
14336 IApptDeviceSeqNum = pAppointmentSequenceNum->GetSequenceNumber();
14337
14338 // Search for state info corresponding to this ref id
14339 hResult = MAPIAllocateBuffer( sizeof( SBinary ), reinterpret_cast<VOID**>( &pbinStateSourceKey ) );
14340 if( CallFailed( hResult ) )
14341 {
14342     ReportError( DebugLog::LOG_WARNING, mailbox_fc, "MAPIAllocateBuffer", hResult );
14343     bResult = false;
14344     goto __end;
14345 } // if
14346 pbinStateSourceKey->lph = 0;
14347 pbinStateSourceKey->cb = 0;
14348 m_CalSyncStateMutex.Lock();
14349 bFound = m_pCalendarState->QueryByRefId( lApptRefId, pbinStateSourceKey,
14350     &lStateParentRefId, &fStateLastModified,
14351     &lStateDeviceSeqNum, &lStateDesktopSeqNum,
14352     &fStateDelInProgress, &fStateModInProgress );
14353 m_CalSyncStateMutex.Unlock();
14354 // If state info exists
14355 if( bFound )
14356 {
14357     // Get item sequence number and compare to see if item needs to be filtered
14358     if( lStateDeviceSeqNum > lApptDeviceSeqNum )
14359     {
14360         // filter
14361         pCalendarEvent = 0;
14362         pAppointment->GetNextProperty( &pCalendarEvent );
14363     }
14364 }

```

```

14365         continue;
14366     }
14367     // If not filtered, use source key (and possibly instance date) to retrieve item
14368     // using MAPI. Get the start and end time to generate a CDO filter.
14369
14370     // Get contents table for calendar folder
14371     hResult = m_pFolders[ CALENDAR ]->GetContentsTable( 0, &pContentsTable );
14372
14373     if( CallFailed( hResult ) )
14374     {
14375         ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "GetContentsTable", hResult );
14376         bResult = false;
14377         goto __end;
14378     } // if
14379
14380     // Set properties we are interested in.
14381     hResult = pContentsTable->SetColumns( reinterpret_cast<SPropTagArray*>( &sptaAppointmentProps ), 0 );
14382     if( CallFailed( hResult ) )
14383     {
14384         ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "SetColumns", hResult );
14385         bResult = false;
14386         goto __end;
14387     } // if
14388     // Restrict the rows returned to only those with a matching Source Key value.
14389     sSourceKeyRes.rt = RES_CONTENT;
14390     sSourceKeyRes.res.resContent.ulFuzzyLevel = FL_FULLSTRING;
14391     sSourceKeyRes.res.resContent.ulPropTag = PR_SOURCE_KEY;
14392     spvSourceKey.ulPropTag = PR_SOURCE_KEY;
14393     spvSourceKey.Value.bin.cb = pbinStateSourceKey->cb;
14394     spvSourceKey.Value.lpb = pbinStateSourceKey->lpb;
14395     sSourceKeyRes.res.resProperty.lpProp = &spvSourceKey;
14396     hResult = pContentsTable->Restrict(&sSourceKeyRes, TBL_BATCH);
14397     if( CallFailed( hResult ) )
14398     {
14399         // Query failed.
14400         ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "Restrict", hResult );
14401         goto __end;
14402     } // if
14403
14404     // Get all matching rows. Should only be one, otherwise error.
14405     hResult = HrQueryAllRows( pContentsTable, 0, 0, 0, &pRows );

```

```

14410 if( CallFailed( hResult ) )
14411 {
14412     ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "HrQueryAllRows", hResult );
14413     goto __end;
14414 } // if
14415 if( pRows->cRows != 1 )
14416 {
14417     // error: should only be one row returned
14418 }
14419 // If this is a root appointment, indicated by no related-to id, use start and end time for filter
14420 if( !ApptParentRefId && !StateParentRefId )
14421 {
14422     fStartTimeFilter = pRows->aRow[ 0 ].lpProps[ 4 ].Value.ft;
14423     fEndTimeFilter = pRows->aRow[ 0 ].lpProps[ 5 ].Value.ft;
14424 } // Else if this is exception
14425 else
14426 {
14427     // Open the root appointment
14428     pRootAppointment = OpenMessage( reinterpret_cast<LPENTRYID>(pRows->aRow[ 0 ].lpProps[ 0 ].Value.bin.lpb),
14429                                     pRows->aRow[ 0 ].lpProps[ 0 ].Value.bin.cb,
14430                                     CALENDAR );
14431 // Get the attachment table for this appointment
14432 hResult = pRootAppointment->GetAttachmentTable( 0, &pAttachTable );
14433 if( CallFailed( hResult ) )
14434 {
14435     ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "GetAttachmentTable", hResult );
14436     bResult = false;
14437     goto __end;
14438 } // if
14439 // Set properties we are interested in.
14440 hResult = pAttachTable->SetColumns( reinterpret_cast<SPropTagArray*>( &sptaAttachProps ), 0 );
14441 if( CallFailed( hResult ) )
14442 {
14443     ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "SetColumns", hResult );
14444     bResult = false;

```

```

14450         goto __end;
        } // if

// *****
// *** Process Attachments ***
// *****

// Process all attachments.
hResult = HrQueryAllRows( pAttachTable, 0, 0, 0, &pAttachRows );
if( CallFailed( hResult ) )
{
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "HrQueryAllRows", hResult );
    bResult = false;
    goto __end;
} // if

// Call GetInstanceFilter to get start & end time for CDO filter, instance is Recurrence-ID
UnixTimeToFileTime( tInstanceDate, fInstanceDate );
GetInstanceFilter( pRootAppointment, pAttachRows, pAttachRows->cRows,
    fInstanceDate, &fStartTimeFilter, &fEndTimeFilter );

14470     }

// Using CDO filter, obtain appt object and update using values contained in the RIMCalendar
GetCDOAppointment( spAppointmentItem, spCalendarFolder,
    fStartTimeFilter, fEndTimeFilter,
    pRows->aRow[0].lpProps[2].Value.I);
}
else
{
    // Else if no state record for this item, then it's a new record
    // If the item is a child appointment, indicated by Related-To property
    if( pAppointmentMethod->GetMethod() == RIMCalendar::CANCEL )
    {
        // No state info, so can't delete it. Skip and go to next event.
    }

    if( lApptParentRefId )
    {
        // Retrieve state info for parent. Will need to identify instance and to update state.
        // Search for state info corresponding to the parent ref id
        hResult = MAPIA::AllocateBuffer( sizeof( SBinary ), reinterpret_cast<VOID**>( &pbinStateSourceKey ) );

```



```

14495         if( CallFailed( hResult ) )
        {
            ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "MAPILocateBuffer", hResult );
            bResult = false;
            goto __end;
        } //if
        m_CalSyncStateMutex.Lock();
        bFound = m_pCalendarState->QueryByRefId( lApptParentRefId, pbmStateSourceKey,
            &lStateParentRefId, &lStateLastModified,
            &lStateDeviceSeqNum, &lStateDesktopSeqNum,
            &lStateDefInProgress, &lStateModInProgress );
        m_CalSyncStateMutex.Unlock();

        // Search for parent item. Table columns will be source key, inet article #, duration, last mod
        // Restrict the rows returned to only those with a matching Source Key value.
        sSourceKeyRes.rt = RES_CONTENT;
        sSourceKeyRes.res.resContent.ulFuzzyLevel = FL_FULLSTRING;
        spvSourceKey.ulPropTag = PR_SOURCE_KEY;
        spvSourceKey.Value.bin.cb = pbmStateSourceKey->cb;
        spvSourceKey.Value.bin.lpb = pbmStateSourceKey->lpb;
        sSourceKeyRes.res.resProperty.lpProp = &spvSourceKey;
        hResult = pContentsTable->Restrict(&sSourceKeyRes, TBL_BATCH);
        if( CallFailed( hResult ) )
        {
            // Query failed.
            ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "Restrict", hResult );
            goto __end;
        } //if

        // Get all matching rows. Should only be one, otherwise error.
        hResult = HrQueryAllRows( pContentsTable, 0, 0, 0, 0, &pRows );
        if( CallFailed( hResult ) )
        {
            ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "HrQueryAllRows", hResult );
            goto __end;
        } //if

        if( pRows->cRows != 1 )
        {
            // error: should only be one row returned

```

```

14535     }
        // Build CDO filter using date/time in Recurrence-ID property, plus duration
        UnixTimeToFileTime(InstanceDate, ftStartTimeFilter);
        UnixTimeToFileTime((InstanceDate+pRows->aRow[0].lpProps[2].Value.l*60), ftStartTimeFilter);
        // Retrieve appt object
        GetCDOAppointment( spAppointmentItem, spCalendarFolder,
            ftStartTimeFilter, ftEndTimeFilter,
            pRows->aRow[0].lpProps[2].Value.l );
    }
    // Else
    else
    {
        // New stand alone appointment. Create appt object.
        spCalendarMessages = spCalendarFolder->GetMessages();
        spAppointmentItem = spCalendarMessages->Add();
    }
}
// If this is a PUBLISH method
if( pAppointmentMethod->GetMethod() == RIMCalendar::PUBLISH )
{
    RIMEventtoCDOAppointment(&CalendarVeent, spAppointmentItem);
    spAppointmentItem->Update(true,true);
}

// Update state info
spAppointmentSourceKey = spAppointmentFields->GetItem((long)PR_SOURCE_KEY);
vtAppointmentSourceKey = spAppointmentSourceKey->GetValue();
vtLastModified = spAppointmentItem->GetTimeLastModified();
if(!VariantTimeToSystemTime(vtLastModified, &stLastModified) ||
    !SystemTimeToFileTime(&stLastModified, &ftLastModified))
{
    DebugLog::Printf( DebugLog::LOG_WARNING,
        "MAPIMailbox::RIMMeetingtoMAPIMeeting - Converting VariantTime to FileTime Failed." );
    bResult = false;
    goto __end;
}

if( !(pbinStateSourceKey->lpb) )
{
    // JAG Note: Redo this whole thing! Uckkk!
    hResult = MAPIAIAllocateMore( strlen( bstr_1(vtAppointmentSourceKey))/2,
        pbinStateSourceKey,
        reinterpret_cast<VOID**>( &(pbinStateSourceKey->lpb) ) );
    if( CallFailed( hResult ) )

```

```

14580 {
    ReportError( DebugLog::LOG_WARNING, mailbox_fcn, "MAPIAIlocateMore", hResult );
    bResult = false;
    goto __end;
} // if
if(!fBinFromHex( bstr_t( vtAppointmentSourceKey ), pbinStateSourceKey->|pb))
{
    //error
}
pbinStateSourceKey->cb = strlen( bstr_t( vtAppointmentSourceKey ))/2;
}
m_CalSyncStateMutex.Lock();
m_pCalendarState->UpdateStateRecord( lApptRefId, pbinStateSourceKey, 0,
    lApptParentRefId ? &fInstanceDate : 0,
    lApptParentRefId,
    &fLastModified,
    lApptDeviceSeqNum, 0, false, false );
m_CalSyncStateMutex.Unlock();
}
// Else if this is a CANCEL method
else if( pAppointmentMethod->GetMethod() == RIMCalendar::CANCEL )
{
    spAppointmentItem->Delete();
    spAppointmentItem = 0;
    // Delete any state info for this item
    m_CalSyncStateMutex.Lock();
    m_pCalendarState->DeleteByRefId( lApptRefId );
    m_CalSyncStateMutex.Unlock();
}
pCalendarEvent = 0;
pAppointment->GetNextProperty( &pCalendarEvent );
} // while
}
catch( ... )
{
    ReportException(mailbox_fcn);
    bResult = false;
    goto __end;
}
14620

```

```

    } // Exception handling
    _end:
    try
    {
        // Cleanup.
        if( pbinStateSourceKey )
        {
            MAPIFreeBuffer( pbinStateSourceKey );
        } // if
        return( bResult );
    }
    catch( ... )
    {
        // Cleanup failed; not critical (we hope).
        ReportException(mailbox_fcn);
        return( bResult );
    } // if
}

#else
// Desktop redirector somehow called calendar method. Error.
DebugLog::Printf( DebugLog::LOG_ERROR,
    "MAPIMailbox::Synchronize - Desktop Redirector accessed calendar method." );
return( false );
#endif
}
MAPIMailbox.cpp

```

Invention Disclosure Form

Page 1 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM02/21/00 6:09

THE PURPOSE OF THIS FORM IS TO:

- bring to Chuck Meyer's attention the development of ideas that are believed to be novel and of value to RIM;
- provide a description of the Invention that establishes an invention date;
- provide RIM's outside patent counsel with a description of the Invention that facilitates processing of a patent application.

Complete this form by typing the information requested. Submit the completed disclosure form with any additional material requested below to Chuck Meyer.

1. **TITLE OF INVENTION:** Enter a short descriptive phrase that will serve as the title of the Invention.

2. DESCRIPTION OF INVENTION: Enter a brief description of the nature and application of the Invention.

This invention will allow “over the air” changes on a PDA calendar to be synchronized with the data on the host calendar. This includes all types of calendar entries, including recurring appointments and exceptions to a recurrence. All operations will be supported: creation, modification, and deletion of appointment entries. Conflicts that occur during synchronization are resolved without user interaction based on a simple [REDACTED] model where an end point is designated the authority in any dispute. [REDACTED] some [REDACTED] in the implementation of the calendar synchronization feature. The process will allow for multiple type calendars. There will also be a “turn off” option to stop the synchronization of calendar events.

3. Identify the projects and products where the Invention may be incorporated.

Blackberry
Proton
Future PDA products

4. The following dates apply to the Invention:

- a) Date marketing began (or is planning to begin) for any products utilizing the Invention: ____/____/____.
- b) Date RIM shipped (or has plans to ship) any product utilizing the Invention: ____/____/____.
- c) Date any written material in which the Invention is described, has been (or will be) distributed:
- d) If the Invention has been described to anyone outside of RIM who has not signed a confidentiality agreement enter date, names, and circumstances surrounding the disclosure: N/A
- e) If the Invention has been used by anyone outside of RIM enter date, names and circumstances: N/A
- f) First recorded a description of the Invention ____ in _____ presently available at _____.
- g) When was a model of the Invention completed: ____/____/____.
- h) When was a photo of the model taken: N/A.
- i) Device or system utilizing the Invention first tested on ____/____/____ by _____ at _____.

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date _____

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date _____

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date _____



Invention Disclosure Form

Page 2 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

5. Describe the concern of problem your Invention addresses.

Currently the device provides a calendar application where users may maintain appointment and event-related information. Information stored on the host and device are distinct, with the synchronization of the two separate databases only occurring as part of a serial synchronization process when the pager is placed in the docking cradle. ~~Without user intervention, this process must be able to synchronize~~ Conflicts can be avoided with ~~the~~; however, this process will use only one step to reduce the number of data transfers between the device and host.

6. Identify any devices or systems known to you that address the same concern or problem. (Also include those developed by RIM and any known patents or publications that address the problem (attach copies).

7. Identify the differences between your Invention and each known device or article described above and describe any impact those differences can have in providing value to _____.

N/A

8. Identify any other unique aspects of your Invention.

9. Provide a detailed description of your Invention. Attach any drawings, circuit diagrams, or other documentation that describes the Invention; where appropriate, apply reference numerals to drawings and use those reference numerals in your detailed description. You must disclose the best mode you know for practicing the Invention.

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date

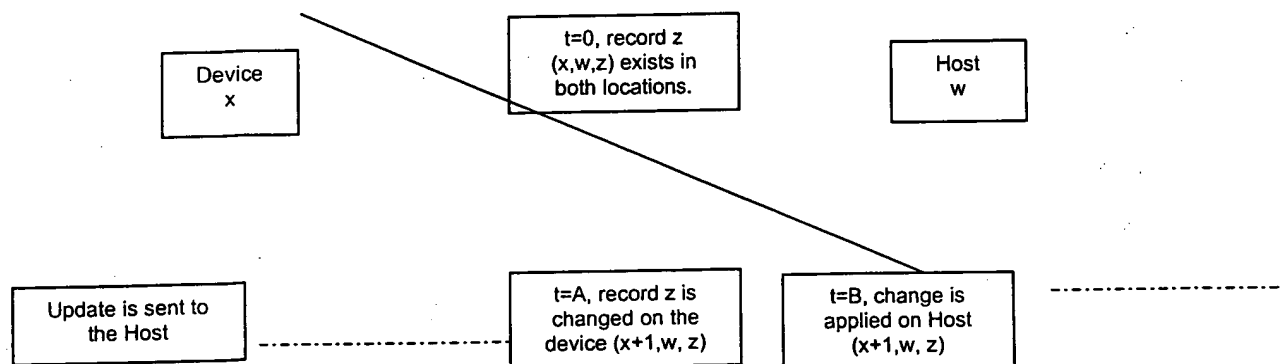


Invention Disclosure Form

Page 3 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

Fig 1: Calendar Record Synchronization for a Device and one Host



Firstly let us consider the case when no conflict occurs. Each record entry has two version numbers: "host version" w and "device version" x . Both the device and host recognise these numbers in the record entry. At $t=0$ (where t is time), the device and host are synchronized and both contain the record which has the version numbers w,x and the record number (e.g. (x,y,w,z)). Suppose the device makes a change. To illustrate, in Fig.1, at $t=A$, the device makes a change to the record z and updates the device version to $x+1$ associated with the record. Preferably, the updated record is sent via the wireless network to the host in order to synchronize the device and host. At $t=B$, the host makes the changes and updates its record z to $x+1$.

Conversely w , when the host makes a change to record z , it updates the host version number to $w+1$ and sends the update via the wireless network to the device at $t=A$. At $t=B$, the device accepts the change and updates its host version number to $w+1$.

A conflict occurs when either the host or device makes a change to a record before $t=B$ (i.e., before notification of the change arrives to the device or host respectively after the change made at $t=A$). The version numbers are now out of sync. In this case one of the changes will be discarded based on the "master" setting which the user selected. This setting selects either the host or the device so that when a conflict arises the "master" will override all changes. No

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This
and

An update conflict occurs when:
A change occurs on the Host ($w+1$) before $t=B$ and arrives to the device after $t=A$ (and vice versa)
No update conflict occurs when:
A change on the Host and an update on the device occur either entirely before $t=A$ or entirely after $t=B$



Invention Disclosure Form

Page 4 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:

03/13/00 4:53 PM 02/21/00 6:09

conflicts occur if the "second" changes occurs on both the device or and host take place entirely before an update occurs, $t < A$ or entirely after the update occurs $t > B$.

Fig 2.1: Conflict when Host is Master
 $t=A$

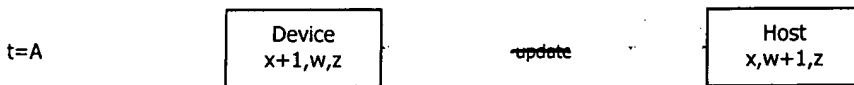


$t=B$



For example, in Fig 2.1, the host is set as the master. At $t=A$ (where t is time), the host makes a change to record z , increments the host version number to $w+1$, and sends an update to the device. However, the device has also made a change to record z , incremented the device version number to $x+1$, and sent an update to the host before receiving the host changes. The two records on the devices are now out of sync. At $t=B$, the host, being the master will discard the device changes. The device upon receipt of the host's update will accept the changes from the host and discard the changes the device made. The device will increment the host version number to $w+1$ and decrement the device version number back down to x .

Fig 2.2: Conflict Resolution when Device is Master



$t=B$



Conversely, in Fig 2.2, the device is set as the master. At $t=A$, both the device and host make an update at the same time, incrementing their respective version numbers to $x+1$ and $w+1$. At $t=B$, the device, being the master, will discard the host's changes. The host, upon receipt of the device's update, will discard the previous change the host made, and accept the device's changes. The host will increment the device version number to $x+1$ and decrement the host version number to w .

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date

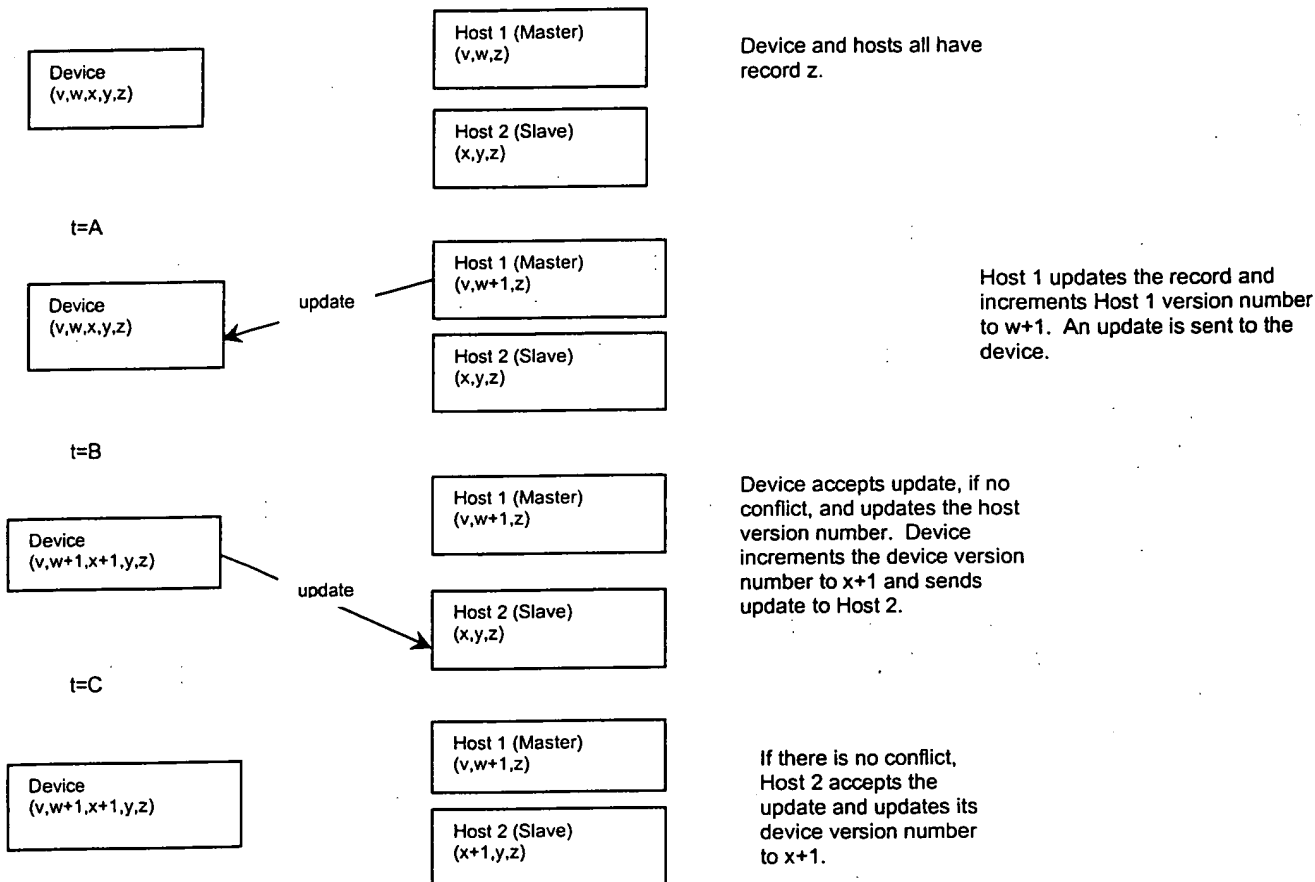


Invention Disclosure Form

Page 5 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

Fig 3: Multiple Hosts - Update sent from Master Host
t=0



The device may also communicate with more than one host (as in Fig. 3). In the preferred embodiment, the hosts do not communicate with each other. In this embodiment, one of the hosts is set as the "master". The device recognizes this setting. The other hosts, or "slave" hosts, recognize the device as their "master".

Indices are scrambled in this paragraph ... In Fig. 3, at t=0 (where t is time), the device and hosts all contain the record. The record is represented by a number z. All hosts have a unique host version number and a unique device version number. The diagram illustrates an example of one device and two-host system. The master host contains the number (v,w,z). The number "w" is the host version number. The master host also has a device version number "v".

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date



Invention Disclosure Form

Page 6 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:

03/13/00 4:53 PM 02/21/00 6:09

The slave host contains the number (x,y,z) . The slave host has a host version number "y" and its own device version number "x". The device contains all these numbers (v,w,x,y,z)

At $t=A$, the master host updates the record z. The master host then increments its version number to $w+1$ and sends an update to the device via the wireless network. At $t=B$, if there is no conflict, the device accepts the change and increments the host version number to $w+1$. The device also increments the device version number that is associated with the slave host to $x+1$ and sends an update to the slave host. At $t=C$, the slave host accepts the change and increments its device version number to $x+1$, as long as there is no conflict.

Fig 4: Multiple Hosts--Update from the Device
 $t=A$

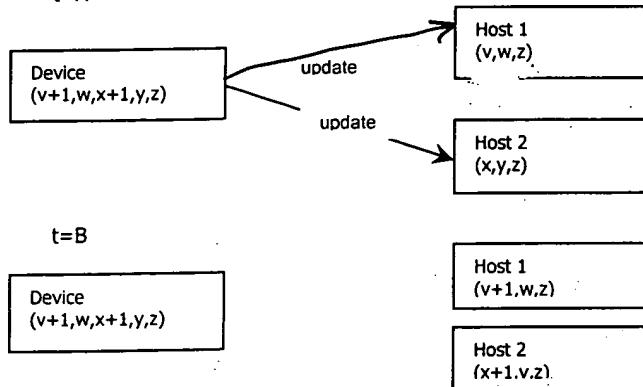
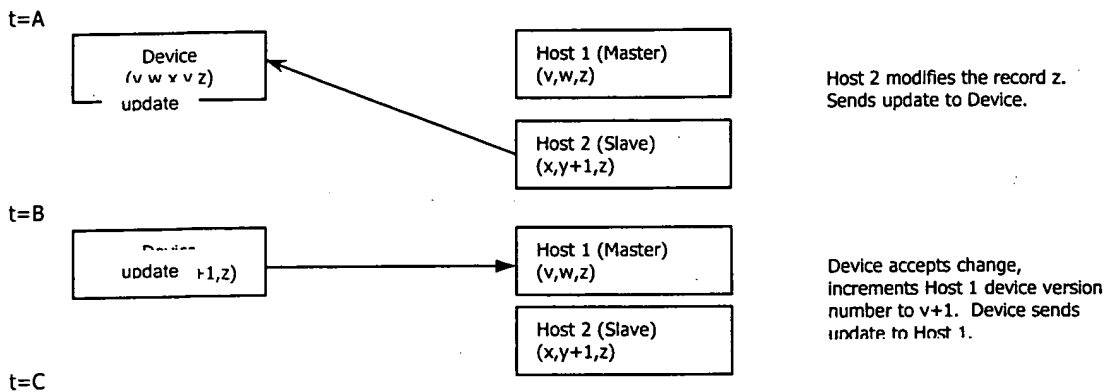


Fig 4 illustrates when the device updates the record z. At $t=A$, it will update the device version numbers to $v+1$ and $x+1$. The device sends an update to both hosts over the wireless network. At $t=B$, if there is no conflict with the hosts, they accept it and update their own device version numbers accordingly. In this situation, the host or the device could be the master.

Fig 5: Multiple Hosts--Update from the Slave Host



Inventors:

Device
(v+1,w,x,y+1,z)

FULL NAME (TYPE)

Craig Dunk

FULL NAME (TYPE)

Host 1 (Master)
(v+1,w,z)

Host 2 (Slave)
(x,y+1,z)

SIGNATURE

Date

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date



Invention Disclosure Form

Page 7 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

Host 1 accepts change

Fig 5 illustrates the situation where the slave host makes an update. At $t=A$ (where t is time), the slave host has changed the record z , increments its host version number y to $y+1$, and sends an update over the wireless network to the device. At $t=B$, the device accepts the modifications and changes the record. The device increments the master host device number from v to $v+1$, and sends an update to the master host. At $t=C$, the master host accepts the change and increments its device version number to $v+1$.

Fig 6: Multiple Hosts -- Conflict Resolution Between Master and Device

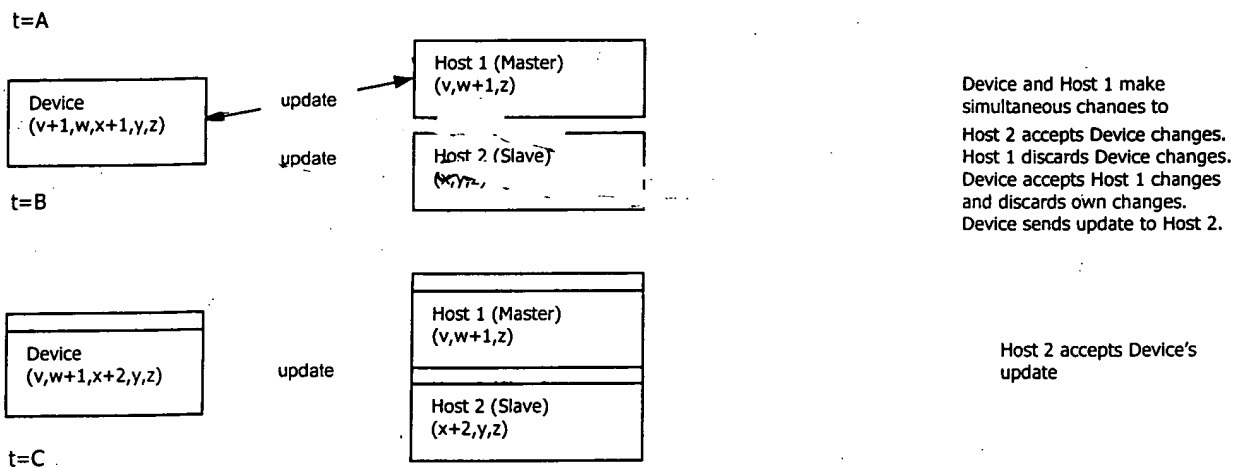


Fig 6 illustrates a conflict resolution in a situation with multiple hosts. At $t=A$ (where t is time), the master host and device make changes to record z . The device increments the device version numbers to $v+1$ (for the master host) and to $x+1$ (for the slave host). The master host increments the master host version number from w to $w+1$. The master host sends an update over the wireless network to the device. The device simultaneously sends an update to the master host and slave host. At $t=B$, the slave host accepts the device changes and increments its device version number to $x+1$. The master host discards the device changes. Upon receipt of the master host changes, the device discards its earlier changes and accepts the master host changes. The device decrements the master host device version number back to v , increments the master host host version number to $w+1$, and increments the slave host device version number to $x+2$. The device sends an update to the slave host. At $t=C$, the slave host accepts the device changes and increments its device version number to $x+2$.

Fig 7: Multiple Hosts -- Conflict Resolution between Slave Host and Device

$t=A$

Inventors:

Hugh Hind
FULL NAME (TYPE)

Craig Dunk
FULL NAME (TYPE)

Host 1 (Master)
(v, w, z)

Host 2 (Slave)
(x, v+1, z)

Device and Host 2 modify record z and send update simultaneously.
Device increments device version numbers $v+1$, $x+1$.

Date

SIGNATURE

Date

This Invention has been witnessed and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date



Invention Disclosure Form

Page 8 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:
03/13/00 4:53 PM 02/21/00 6:09

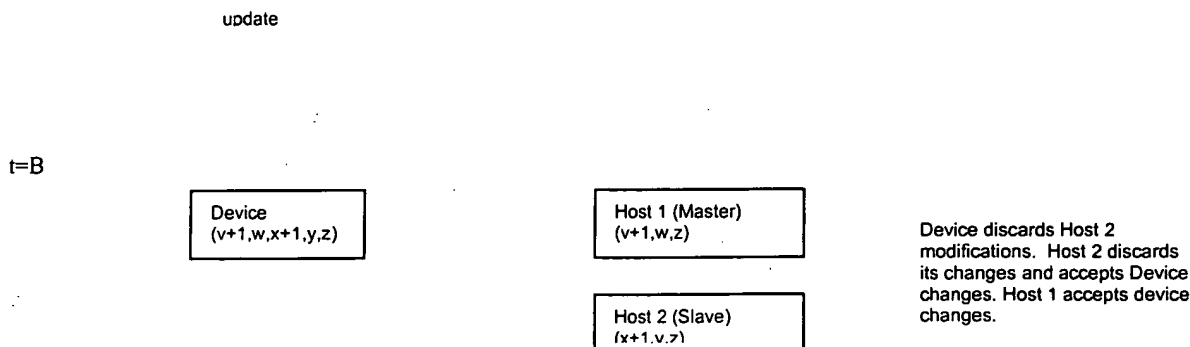


Fig 7 illustrates a conflict resolution between the device and slave host when the slave host sends out an update. At t=A, the slave host changes the record z and increments its host version number to y+1. The device also changes the record and increments the device numbers for the master and slave hosts to v+1 and x+1 respectively. Both the slave host and device send an update via the wireless network simultaneously. There now exists a conflict between the record that the device holds and the update that was sent to the device. Because the device acts as a "master" to the slave host, at t=B, the device discards the changes that the slave host made. An update is sent from the device to the slave host and the slave host discards the previous update and accepts the device's changes. The device sends an update to the master host. The master host will make the necessary updates as long as there is no conflict. The slave host decrements the host version number back to y and increments the device version number to x+1. The device then sends an update to the master host. The master host will make the necessary updates as long as there is no conflict.

Fig 7.2: Multiple Hosts --Conflict between Device and Master Host after Slave Host has sent an update.

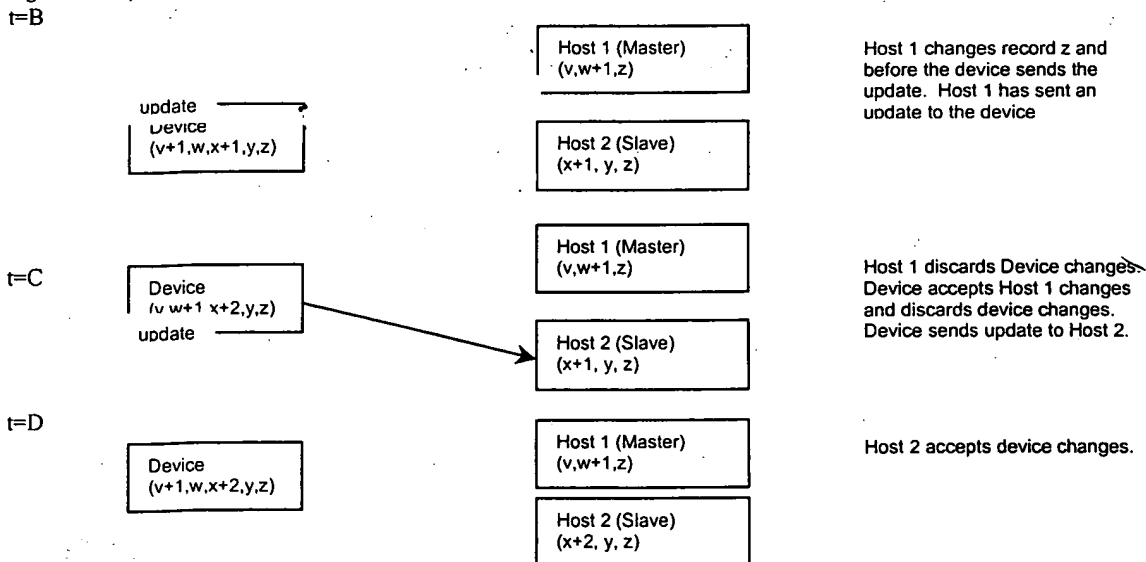


Fig 7.2 continues from Fig 7.1. At t=B, if the master host changes the record (and increments its host version number to w+1) before it receives the device's update, there is now a conflict. At t=C, the master host will discard the update from the device. The device also discards the changes that it made and accepts the master host's update. The device decrements the master host device

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date



Invention Disclosure Form

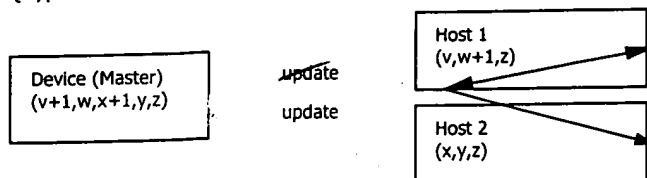
Page 9 of 9
DISCLOSURE DATE:

Reference Number:
TO BE ASSIGNED BY:

03/13/00 4:53 PM 02/21/00 6:09

version number back to v , and increments the slave host device version number to $x+2$. The device sends an update to the slave host. The slave host accepts the update and increments its device version number.

Fig. 8: Multiple Hosts – Conflict Resolution when the Device is Master
 $t=A$



Device makes changes to record and sends update to Host 1 and Host 2. Host 1 also makes changes and sends an update to Device.

$t=B$



Device discards Host 1 changes. Host 1 discards Host 1 changes and accepts Device changes. Host 2 accepts Device changes.

Fig 8 illustrates the situation where the device has the “master” setting for all the hosts and has a conflict with either host. At $t=A$, the device makes a change to record z . The device increments the device version numbers for Host 1 and Host 2 to $v+1$ and $x+1$ respectively. Then the device sends an update to the hosts over the wireless network. Host 1 makes changes to record z and increments its host version number to $w+1$. Host 1 then sends out an update at the same time as the device or before Host 1 has received the device’s update. The host 1 record and the device record are out of sync. At $t=B$, host 2 accepts the Device changes and increments its device version number to $x+1$. The device, because it is set as the master, discards Host 1’s changes. Host 1 accepts the device changes and discards its own changes. Host 1 increments the device version number to $v+1$ and decrements its host version number to w .

Inventors:

Hugh Hind

FULL NAME (TYPE)

SIGNATURE

Date

Craig Dunk

FULL NAME (TYPE)

SIGNATURE

Date

This Invention has been witnessed
and understood by me:

FULL NAME (TYPE)

SIGNATURE

Date